Predicting Piano Fingerings with Machine Learning: A Comparison of Random Forest, RNN, and LSTM

ANASTASIA ISKALIYEVA
Dept Information and Computer Sciences
a.iskaliyeva@gmail.com

ABSTRACT

As a pianist, I often struggle with determining the right finger placement when learning new music, especially for complex pieces and composers like Bach. While some scores include finger annotations, many do not, leaving pianists to either manually annotate them - a time-consuming process - or search for existing annotations on specialized forums. I have personally spent hours annotating fingerings by hand, wishing for a more efficient and accessible solution.

With my growing expertise in machine learning, I had the idea of automating fingering prediction using machine learning to provide a valuable tool for learners, educators, and composers. I developed a model that could be trained on existing annotated sheet music to predict fingerings for new pieces. In this study, I explore the effectiveness of different machine learning models - Random Forest, Recurrent Neural Networks (RNNs), and Long Short-Term Memory (LSTM) networks - for this task.

The dataset consists of real-world annotated piano scores in MusicXML (MXL) format, which I used to train and evaluate the models. The goal was to determine the most effective algorithm for predicting fingerings on unseen musical scores. My results show that the Random Forest algorithm outperforms deep learning methods in generating efficient fingerings with 86% accuracy. These findings suggest that machine learning can be a valuable tool for pianists, reducing the time spent on manual annotation and improving access to high-quality fingering recommendations.

DATA EXPLORATION

The most challenging aspect of the project was data preparation, as music score data has nested structure and complex syntax, which makes it difficult to understand, parse, and manipulate. The process involved parsing publicly available MusicXML files with fingering annotations using the music21 Python library (Figure 1). The dataset was sourced from publicly available piano scores on Musescore.com, focusing on those that included fingering annotations. Key musical features—such as note pitch (specific note in a particular octave), chord structure, rests, and durations—were extracted and compiled into a structured CSV dataset (training.csv) for model training (Figure 2).

Figure 1. Music Data Representation in MusicXML Format

```
<note>
  <pitch>
    <step>E</step>
    <alter>-1</alter>
    <octave>4</octave>
  </pitch>
  <duration>2</duration>
  <type>half</type>
</note>
```

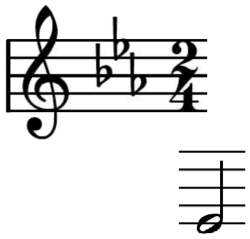


Figure taken from [Meinard Müller, Fundamentals of Music Processing, Figure 4.2, Springer 2015]

```python
from music21 import converter, stream, note, chord, articulations
import pandas as pd

# Function to parse score and extract data into dataset
def extract_fingering_to_dataframe(file):
    score = converter.parse(file)
    data = []
    for part_index, part in enumerate(score.parts):
        for n in part.recurse():
            if isinstance(n, note.Note):
                fingers = [
                    art.fingerNumber for art in n.articulations
                    if isinstance(art, articulations.Fingering)]
                data.append({
                    "title": file,
                    "note": n.nameWithOctave,
                    "pitch": n.pitch.midi, # MIDI number of the pitch
                    "fingering": fingers[0] if fingers else 0,
                    "is_chord": False,
                    "is_rest": False,
                    "hand": part_index
                })
    df = pd.DataFrame(data)
    return df
```

In total, 43 piano scores were processed (1), comprising over 39,000 notes, of which 16,000 included fingering annotations. The data extraction captured score for both hands (right and left), ensuring wide representation of fingering patterns. As a result, I created the dataset that provides a foundation for training machine learning models to predict optimal fingerings for unannotated pieces.

Figure 2. Music Data Representation in CSV Format

title	note	pitch	fingering	is_chord	is_rest	hand	finger
Sinfonia_No._3_-_Johann_Sebastian_Bach.mxl	Rest	0	0	FALSE	TRUE	0	0
Sinfonia_No._3_-_Johann_Sebastian_Bach.mxl	F#5	78	3	FALSE	FALSE	0	3
Sinfonia_No._3_-_Johann_Sebastian_Bach.mxl	G5	79	0	FALSE	FALSE	0	0
Sinfonia_No._3_-_Johann_Sebastian_Bach.mxl	A5	81	0	FALSE	FALSE	0	0
Sinfonia_No._3_-_Johann_Sebastian_Bach.mxl	C5	72	2	FALSE	FALSE	0	2
Sinfonia_No._3_-_Johann_Sebastian_Bach.mxl	B4	71	0	FALSE	FALSE	0	0
Sinfonia_No._3_-_Johann_Sebastian_Bach.mxl	E5	76	0	FALSE	FALSE	0	0
Sinfonia_No._3_-_Johann_Sebastian_Bach.mxl	F#5	78	0	FALSE	FALSE	0	0
Sinfonia_No._3_-_Johann_Sebastian_Bach.mxl	G5	79	0	FALSE	FALSE	0	0
Sinfonia_No._3_-_Johann_Sebastian_Bach.mxl	B4	71	2	FALSE	FALSE	0	2
Sinfonia_No._3_-_Johann_Sebastian_Bach.mxl	A4	69	0	FALSE	FALSE	0	0
Sinfonia_No._3_-_Johann_Sebastian_Bach.mxl	D5	74	3	FALSE	FALSE	0	3
Sinfonia_No._3_-_Johann_Sebastian_Bach.mxl	E5	76	0	FALSE	FALSE	0	0

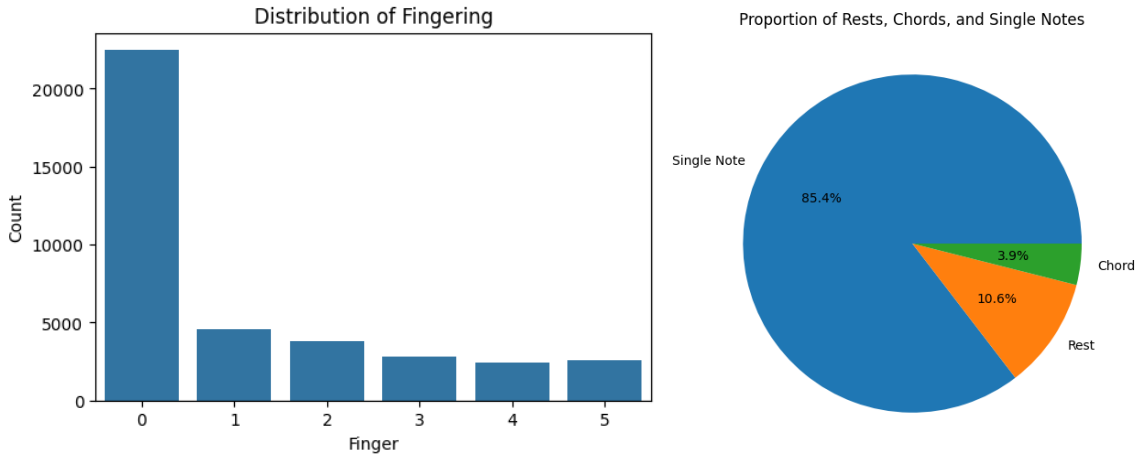
Figure generated by the author

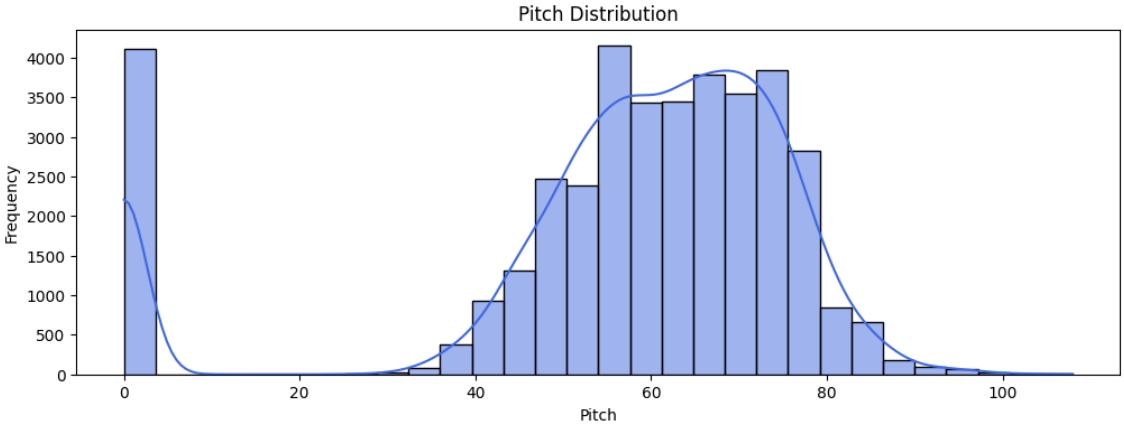
Only 3.9% of samples represent chords, which is reflected in the feature importance of the Random Forest Model, where the "is_chord" feature has the lowest importance score.

The "4" and "5" fingers are least frequently used in the dataset, and all models struggle to differentiate between "3" and "4" fingers consistently.

In order to make notes interpretable by the models I am using note's MIDI number (e.g. "79" for "G5"). Note, the "0" MIDI number ("pitch" column) in our dataset represents the "rest". See Figure 3. for more insights into the dataset.

Figure 3. Distribution of Fingering and Notes



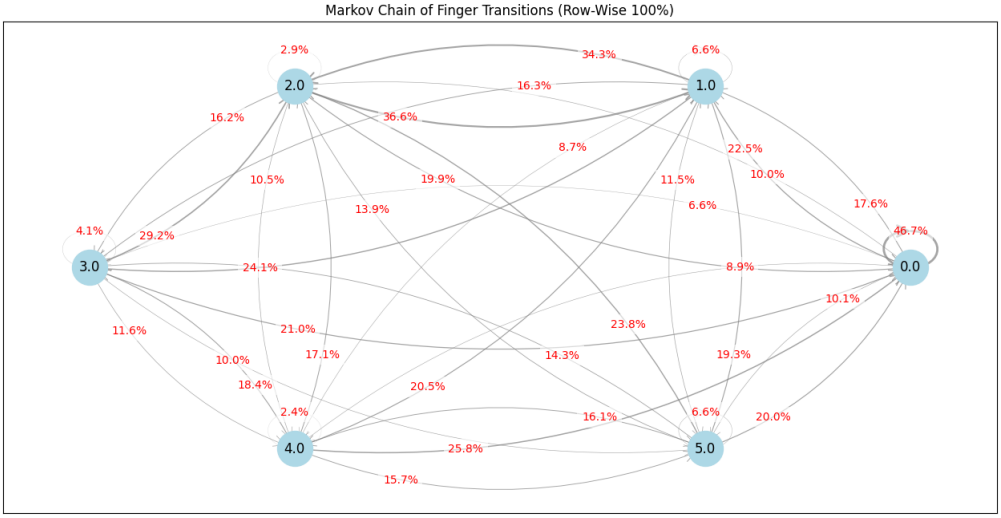


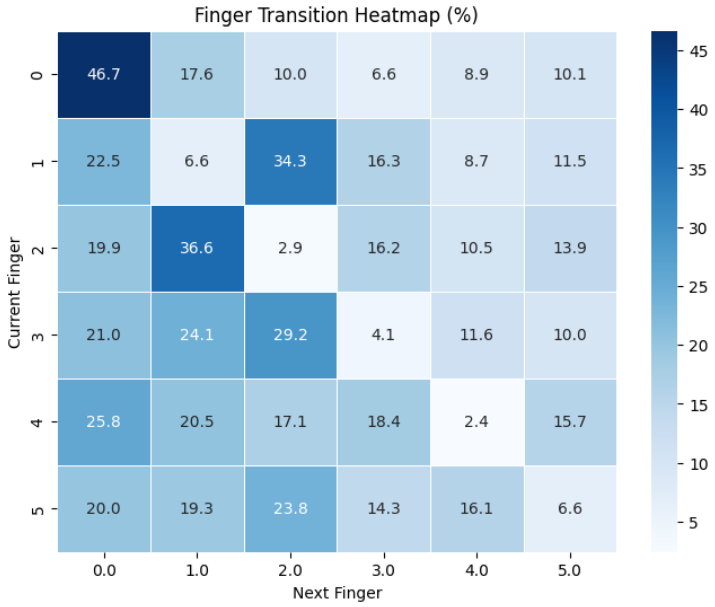
Figures generated by the author

Additionally, I explored transitions between fingers in the annotated scores:

- "0" represents notes that are not annotated with the finger.
- It is uncommon to use the same finger for the next note.
- Most likely transitions are "1 -> 2" and "2 -> 1"

Figure 4. Finger transition charts





MODEL EXPLORATION

Predicting piano fingerings is a classification problem, where each note is assigned a label corresponding to one of 5 possible finger choices (5 fingers per hand). Given the complexity of musical patterns, we need models that can handle structured and sequential data effectively. Traditional machine learning models, like Random Forest, make effective independent predictions for each note fingering, whereas RNNs can leverage temporal dependencies, making them well-suited for capturing musical sequences.

Music notation, which uses symbols like notes, clefs, and duration, visually represents music, much like how written language represents spoken language. Just as words in a sentence have contextual dependencies, musical notes follow specific patterns and phrasing. This means that finger placement depends not just on the current note but also on past and future notes. This inherent sequential nature makes Recurrent Neural Networks a natural choice for modeling piano fingerings, as they are designed to learn from sequential data, maintaining a hidden state that retains memory of past finger choices.

However, standard RNNs suffer from the vanishing gradient problem, making them less effective for capturing long-term dependencies in complex pieces. This is where Long Short-Term Memory (LSTM) networks theoretically improve upon standard RNNs, as they maintain relevant information over longer sequences.

To sum up, in the project I compare three models:

1. Random Forest – a baseline that learns from independent features
2. Recurrent Neural Networks (RNN) – for sequential learning, capturing dependencies over time

3. Long Short-Term Memory (LSTM) Networks – an advanced variant of RNNs that helps preserve long-term dependencies

RANDOM FOREST

The Random Forest Model can not handle sequential data out of the box, so the following features were added explicitly to the dataset:

- “Prev Pitch”: distances with the (4) **previous** notes.
- “Prev Finger”: finger annotations on the (4) **previous** notes.
- “Next Pitch”: distances with the (2) **following** notes.
- “Next Finger”: finger annotations on the (2) **following** notes.

```
# Adding prev notes
df['prev_pitch'] = df.groupby(['title', 'hand'])['pitch'].shift(1).fillna(-1)
df['prev_pitch'] = df['pitch'] - df['prev_pitch']
df['prev_finger'] = df.groupby(['title', 'hand'])['finger'].shift(1).fillna(-1)
#... More shifts here ...

# Adding next notes
df['next_pitch'] = df.groupby(['title', 'hand'])['pitch'].shift(-1).fillna(-1)
df['next_pitch'] = df['next_pitch'] - df['pitch']
df['next_finger'] = df.groupby(['title', 'hand'])['finger'].shift(-1).fillna(-1)
```

Since predicting "0" (zeros) makes no sense for the application, I removed all samples that have no finger annotations.

The Random Forest Model has following parameters:

- 'n_estimators' - the number of trees in the forest.
- 'max_depth' - the maximum depth of the tree.
- 'min_samples_split' - the minimum number of samples required to split.
- 'min_samples_leaf' - the minimum number of samples at a leaf node.

I initiated a grid search with some common parameters in order to find the set of parameters that deliver highest accuracy:

```
# Define hyperparameter grid
param_grid = {
    'n_estimators': [50, 150],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [2, 4, 8]
}
# Initialize the RandomForestClassifier
```

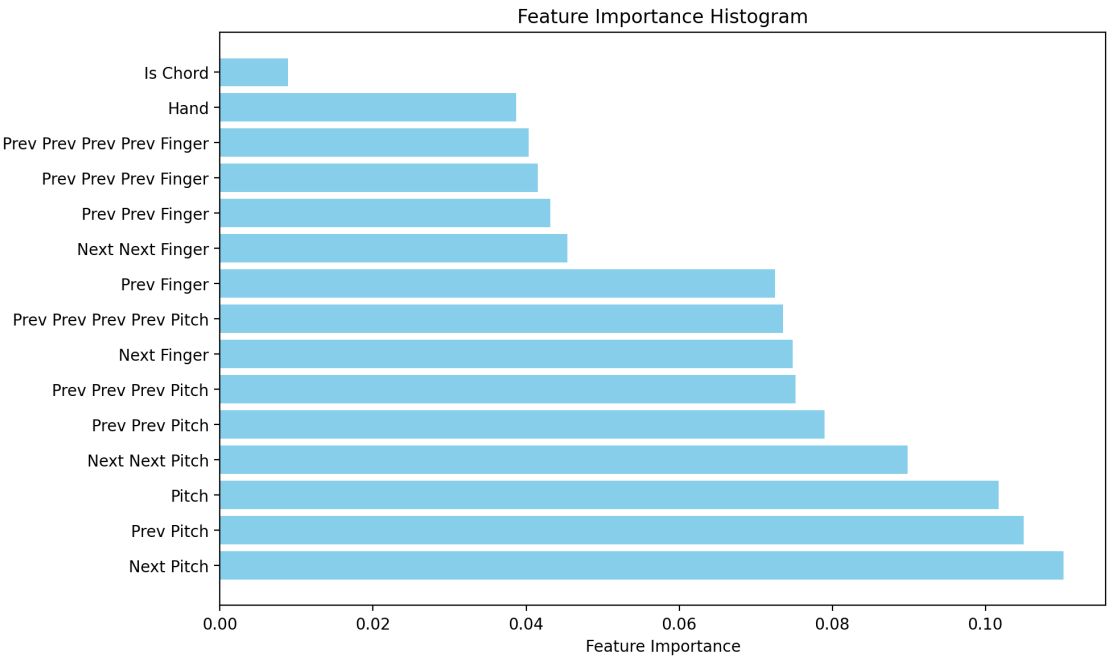
```
rf = RandomForestClassifier(random_state=42)
# Perform grid search with cross-validation
grid_search = GridSearchCV(rf, param_grid, cv=3, scoring='accuracy')
grid_search.fit(X_train, y_train)
```

The selected model has mean cross-validated score of 81% (accuracy), and the following parameters:

- 'n_estimators': 150
- 'max_depth': None,
- 'min_samples_leaf': 2
- 'min_samples_split': 2

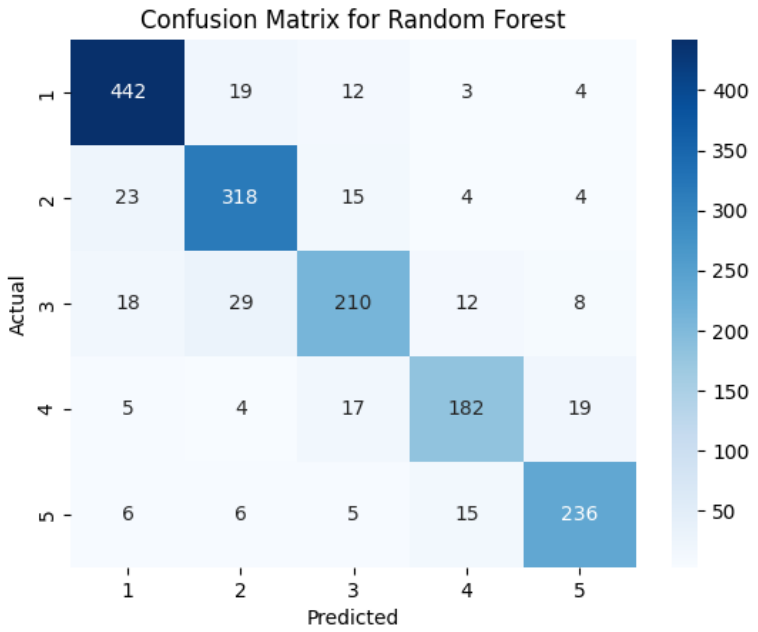
Some models in the search scored as low as 69% (accuracy).

The feature importance is relatively stable across various model versions:



As expected, the distance to the previous and the next (following) notes are the most important features and the hand and the chord are the least important. That is, finger preference is driven by the distances between notes, regardless of whether they are played simultaneously or sequentially, by left hand or right.

The test set **accuracy is 86%**. Confusion matrix is dominated by its diagonal, meaning in most cases the model predicts the right finger. The "3" and "4" are the most confused fingers.



RNN

Unlike the Random Forest Model, the RNN and its variant LSTM can handle sequential data by design, so no additional features were added to the original dataset. But the neural network require data normalization, so I scaled the pitch column:

```
df['pitch'] = (df['pitch'] - df['pitch'].mean()) / df['pitch'].std()
```

For RNN/LSTM models I am predicting the **5th finger in a sequence of 7 fingers** to replicate the dataset generated for the Random Forest Classifier:

```
# Define the dataset class
class PianoFingeringDataset(Dataset):
   def __init__(self, df, seq_length=7):
       self.seq_length = seq_length
       self.features = df[['pitch', 'is_chord', 'is_rest',
'hand']].values.astype(np.float32)
       self.targets = df['finger'].values.astype(np.int64)

   def __len__(self):
       return len(self.features) - self.seq_length

   def __getitem__(self, idx):
       x = self.features[idx:idx + self.seq_length]
       # Predict the 5th element's finger
       y = self.targets[idx + self.seq_length - 3]
```

```
        return torch.tensor(x), torch.tensor(y)
```

Since only part of the dataset has notes annotated with finger numbers, I am using the masking technique, in order to model to focus on important samples only:

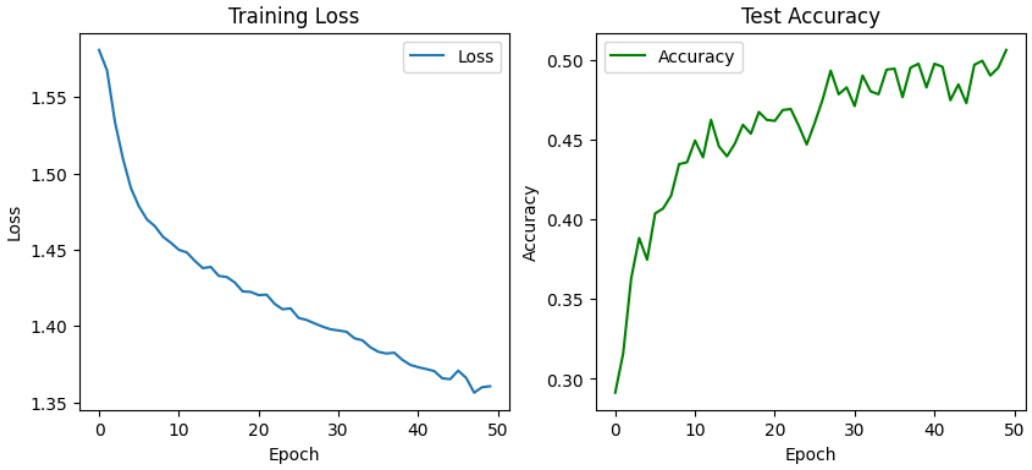
```
# Custom masked loss function
def masked_loss(output, target, criterion):
    mask = target > 0  # Ignore cases where target == 0
    target = target - 1  # Shift range [1-5] → [0-4] for CrossEntropyLoss
    loss = criterion(output[mask], target[mask])
    return loss

# Function to compute accuracy wrt missing values
def compute_accuracy(model, dataloader):
    model.eval()
    correct, total = 0, 0
    with torch.no_grad():
        for x, y in dataloader:
            x, y = x.to(device), y.to(device)
            output = model(x)
            predictions = torch.argmax(output, dim=1) + 1  # Shift [0-4] back to
[1-5]
            mask = y > 0  # Ignore unknown fingers (0)
            correct += (predictions[mask] == y[mask]).sum().item()
            total += mask.sum().item()
    return correct / total if total > 0 else 0
```

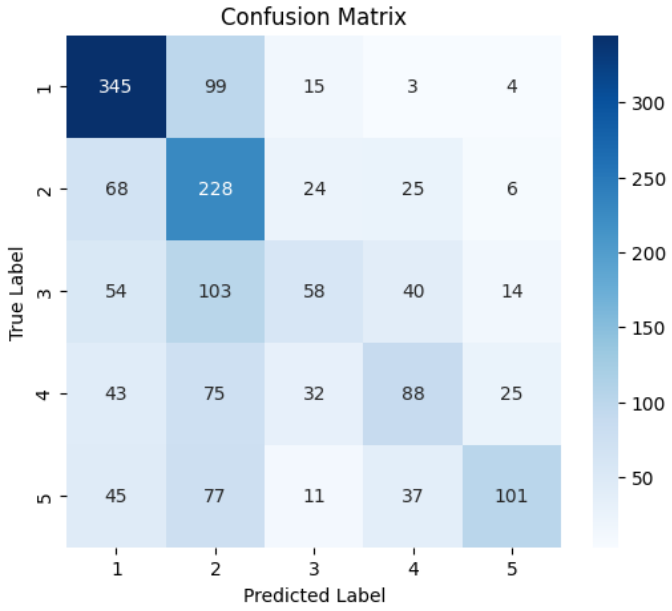
Finally I define the RNN Finger Classifier with 128 features in the hidden state.

```
class RNNFingerClassifier(nn.Module):
    def __init__(self, input_size=4, hidden_size=128, output_size=5):
        super(RNNFingerClassifier, self).__init__()
        self.hidden_size = hidden_size
        self.rnn = nn.RNN(input_size, hidden_size, batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)
        self.softmax = nn.Softmax(dim=1)

    def forward(self, x):
        h0 = torch.zeros(1, x.size(0), self.hidden_size).to(x.device)
        out, _ = self.rnn(x, h0)
        out = out[:, -1, :]
```

```
        out = self.fc(out)
        out = self.softmax(out)
        return out
```

The original model achieved only 51% accuracy after 50 epochs with a learning rate of 0.001.



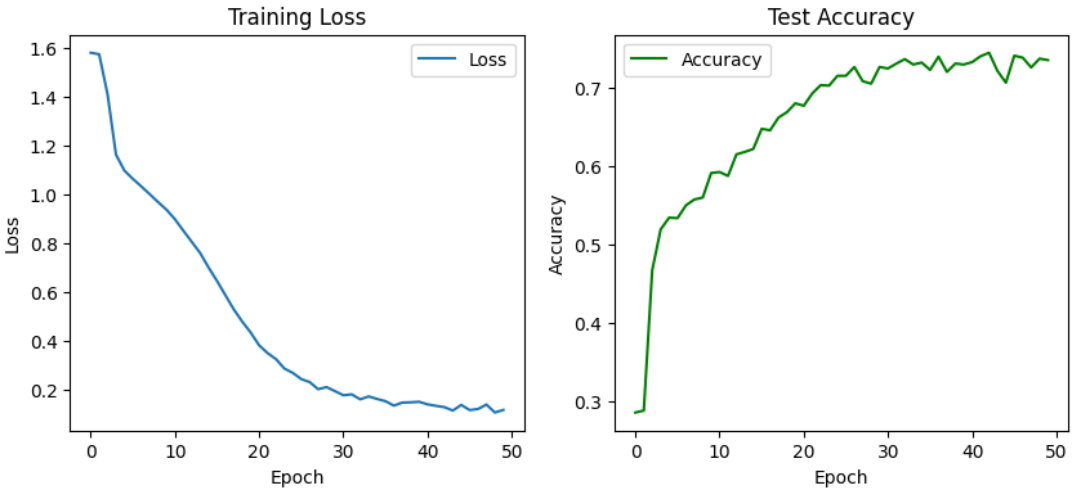


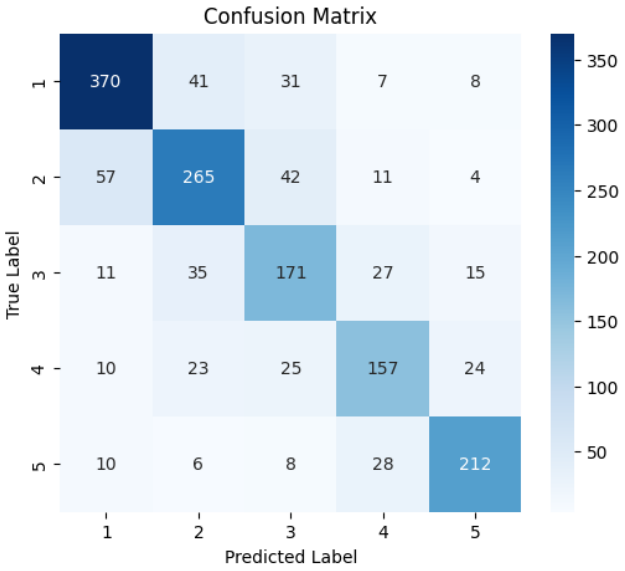
In an attempt to enhance model performance, I added additional features, as RNNs leverage sequential information. Additionally, I assigned weights to the fingers based on expert opinions and personal experience, with the assumption that fingers 2 and 3 are used more frequently, while the others are less often used due to their weaker strength. However, these adjustments did not improve the model's accuracy.

LSTM

Lastly, I implemented an LSTM model in an attempt to capture long-term dependencies in music scores. The best result achieved was accuracy of 73% (measured on the test set).

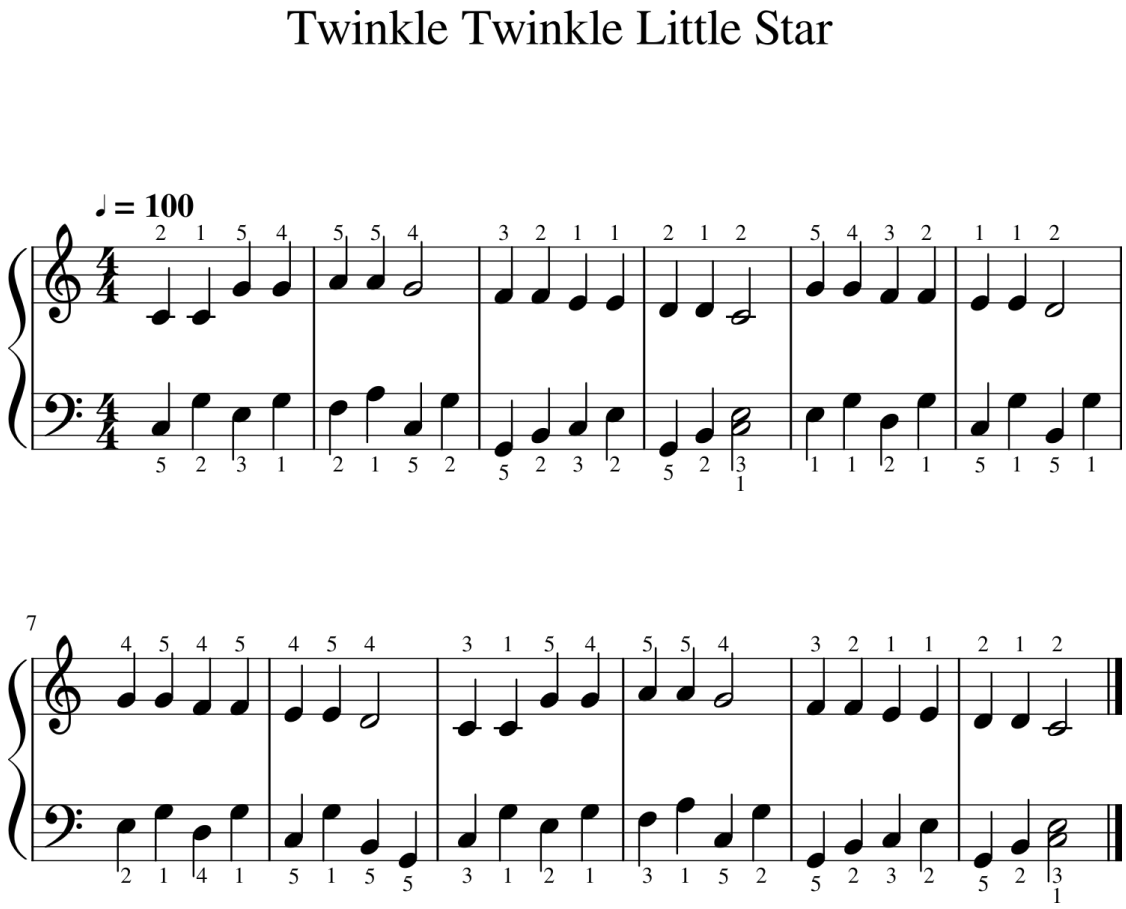
```
class LSTMFingerClassifier(nn.Module):
    def __init__(self, input_size=4, hidden_size=128, num_layers=4, output_size=5):
        super(LSTMFingerClassifier, self).__init__()
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        _, (hn, _) = self.lstm(x)
        out = self.fc(hn[-1])
        return out
```





APPLICATION

In order to make the models actionable and applicable, I have created a Python script that annotates music scores with model provided classifications.



CONCLUSION

Through model comparison, I analyzed how different machine learning techniques handle piano fingering prediction. Random Forest demonstrated strong performance due to its feature-driven approach, while RNNs showed potential in capturing longer-term dependencies. Further improvements may be achieved by combining traditional machine learning and deep learning techniques to enhance accuracy.

This project merges my passion for piano with my data science skills, and I hope it contributes to the learning experience for musicians facing similar challenges.

For more details, please visit my GitHub repository:
https://github.com/Anastasia1707/piano-fingering.